Downloaded from orbit.dtu.dk on: May 05, 2023

DTU Library

An adaptive large neighborhood search procedure applied to the dynamic patient admission scheduling problem

Lusby, Richard Martin ; Schwierz, Martin; Range, Troels Martin; Larsen, Jesper

Published in:
Artificial Intelligence in Medicine

Link to article, DOI:
10.1016/j.artmed.2016.10.002

Publication date:
2016

Document Version
Peer reviewed version

Link back to DTU Orbit

Citation (APA):
Lusby, R. M., Schwierz, M., Range, T. M., & Larsen, J. (2016). An adaptive large neighborhood search procedure applied to the dynamic patient admission scheduling problem. *Artificial Intelligence in Medicine*, *74*, 21-31. https://doi.org/10.1016/j.artmed.2016.10.002

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

An adaptive large neighborhood search procedure applied to the dynamic patient admission scheduling problem

Richard Martin Lusby[a,1], Martin Schwierz[b], Troels Martin Range[c], Jesper Larsen[a]

[a] *Department of Engineering Management, Produktionstorvet, Technical University of Denmark, Kongens Lyngby, 2800, Denmark*
[b] *AMCS Denmark A/S, Hejrevej 34D, Copenhagen, 2400, Denmark*
[c] *Hospital of South West Jutland, Finsensgade 35, 6700, Esbjerg, Denmark*

Abstract

Objective: The aim of this paper is to provide an improved method for solving the so-called dynamic patient admission scheduling (DPAS) problem. This is a complex scheduling problem that involves assigning a set of patients to hospital beds over a given time horizon in such a way that several quality measures reflecting patient comfort and treatment efficiency are maximized. Consideration must be given to uncertainty in the length of stays of patients as well as the possibility of emergency patients.
Method: We develop an adaptive large neighborhood search (ALNS) procedure to solve the problem. This procedure utilizes a Simulated Annealing framework.
Results: We thoroughly test the performance of the proposed ALNS approach on a set of 450 publicly available problem instances. A comparison with the current state-of-the-art indicates that the proposed methodology provides solutions that are of comparable quality for small and medium sized instances (up to 1000 patients); the two approaches provide solutions that differ in quality by approximately 1% on average. The ALNS procedure does, however, provide solutions in a much shorter time frame. On larger instances (between 1000-4000 patients) the improvement in solution quality by the ALNS procedure is substantial, approximately 3-14% on average, and as much as 22% on a single instance. The time taken to find such results is, however, in the worst case, a factor 12 longer on average than the time limit which is granted to the current state-of-the-art.
Conclusion: The proposed ALNS procedure is an efficient and flexible method for solving the DPAS problem.

Keywords: Metaheuristic, adaptive large neighborhood search, patient admission scheduling
2010 MSC: 90B50, 90C27

1. Introduction

Providing high quality health care is one of the most important challenges worldwide. To efficiently, and perhaps even safely, manage the day-to-day activities of a hospital requires a thorough understanding of the hospital's patient flow. In many countries, the admission of patients

*Corresponding author
Email address: rmlu@dtu.dk (Richard Martin Lusby)

to hospitals is administered from a central unit, its responsibility being to facilitate the best possible assignment of patients to beds in the hospital. This is indeed a non-trivial task. No two patients are identical. Each patient's expected length of stay at the hospital, not to mention the specific treatment they require, is likely to be different. Furthermore, in addition to having a finite capacity, a hospital only has a finite set of resources with which all required services can be performed. Coordinating the diverse set of patient requests with the hospital's available resources is thus an important, combinatorial optimization problem and is termed the patient admission scheduling (PAS) problem. Despite its complexity, however, the problem, is typically still solved manually by an experienced nurse or planner.

Patient admission scheduling is a problem that arises at all planning levels within a hospital. On a strategic (i.e. long term) planning, level, historical data is used to forecast the future arrival patterns of patients and determine whether or not the hospital's available resources and their configuration will perform satisfactorily. Examples of research addressing strategic level variants can be found in Hutzschenreuter et al. [1] and Chen et al. [2]. The former presents a simulation tool that is used to find good, long term policies, where the aim is to maximize the yearly throughput of the hospital. The latter develops a genetic algorithm for an admission scheduling problem for a single department. This algorithm exploits historical data and optimizes a long-term admission strategy. Other examples include, among others, Kusters and Groot [3], Harper [4], and Jittamai and Kangwansura [5].

The PAS problem also arises at the tactical planning level. This variant of the problem considers a medium to long term planning horizon, i.e. from a few weeks to some months ahead of time, and attempts to assign upcoming patients to beds in such a way that several quality measures reflecting patient comfort and treatment efficiency are maximized. Almost always is the data assumed to be known with certainty, i.e. a *static* offline problem is solved. As such, it typically only considers elective patients, whose admission and discharge dates are stated exactly; doctors can, in the absence of complications, usually state with a high degree of accuracy the length of stay needed by a patient for a given treatment. Emergency patients, i.e. those that arrive unexpectedly, are normally implicitly considered by simply reserving a certain number of beds. The work by Demeester et al. [6] formalized an academic version of this problem, and was accompanied with 14 publicly available data sets, which are based on real-life instances of the problem. The authors proposed a hybrid tabu search algorithm to solve this problem. The same instances have subsequently been considered by Ceschia and Schaerf [7], Bilgin et al. [8], and Range et al. [9]. The first two contributions also consider local search based heuristics, while the last describes a column generation based heuristic approach combined with constraint aggregation. The general consensus is that exact methods are only tractable for small problem instances (a two week horizon with around at most 150 rooms and 780 patients).

Due to the dynamic setting in which hospitals operate, it is extremely unlikely that a tactical level plan will be executed unchanged in practice. On an operational level, modifications to planned schedules may be required whenever unforeseen events occur. In an effort to better reflect the real-life characteristics of the problem, Vancroonenburg et al. [10], and Ceschia and Schaerf [11] have independently introduced variants of the tactical level problem which attempt to consider operational level uncertainty; these variants are collectively known as the dynamic patient admission scheduling (DPAS) problem *under uncertainty* (henceforth referred to as simply

the DPAS problem). Both studies extend the static PAS problem presented in Demeester et al. [6] in similar ways and attempt to capture several practical uncertainties encountered in the day-to-day running of a hospital. In particular, the assumption that all patient admission dates are known a priori is no longer made; these dates sporadically reveal themselves over time. In other words, associated with each patient is a *registration date*. This is the time at which the patient's planned admission date becomes known to the hospital and it does not necessarily need to be the start of the planning horizon. Other dynamic extensions include the possibility to delay a patient's expected admission date, the possibility for patients to stay longer than expected, and emergency patients. Ceschia and Schaerf [11] test and compare an exact integer linear programming (ILP) formulation, solved using the commercial solver Cplex, and a simulated annealing (SA) approach on a set of 450 self-generated instances, which have since been made public. Results suggest the exact approach is only tractable for small instances of the problem, while the metaheuristic approach can at least find feasible solutions for all instance sizes. As no comparison with the exact approach is possible for the larger instances, and as no bound from the ILP's linear programming relaxation is considered, it is difficult for the authors to vouch for the quality of the solutions obtained in these cases. Vancroonenburg et al. [10] present two ILPs for dynamic versions of the six smallest data sets given in Demeester et al. [6], and consider the impact of emergency patients and patient length of stay estimates. More recently, Ceschia and Schaerf [12] have extended the work of Ceschia and Schaerf [11] to also consider the scheduling of operating theaters.

In this paper we also focus on the DPAS problem and devise an adaptive search (ALNS) procedure embedded within a simulated annealing (SA) framework to solve it. We concern ourselves with the variant proposed by Ceschia and Schaerf [11]. While this problem may not perfectly reflect reality, we believe it has many aspects that are consistent with what is found in practice. We are therefore motivated to not only develop algorithms that can quickly provide good solutions to these problems, but also to, through the algorithmic development, gradually solve larger and more realistic data sets. Currently there is definitely a difference in what can be solved in the academic world and what can be implemented in practice. There is therefore exciting potential in the application of Operations Research techniques to PAS problems, and we hope to inspire others by researching in this area. In this work we focus on providing a tool that can be used on a tactical planning level, where information about patients is gradually revealed over time.

We test and compare the proposed methodology on the large set of instances from Ceschia and Schaerf [11]. Through a direct comparison with their algorithm, not only can we say something about the performance of our algorithm, but we can also comment on the quality of the solutions found in Ceschia and Schaerf [11]. An exhaustive computational study on the 450 available instances shows that the proposed methodology is extremely competitive. On small instances solutions of similar quality are obtained, much faster. On medium to large instances our approach is consistently better. For the large instances the difference in solution quality is sometimes substantial; the algorithm finds better solutions in all cases, and is, on average, 3-14% better. However, the time needed to find such solutions is significantly longer, a factor 12 on average longer than the time limit which is granted to the algorithm in Ceschia and Schaerf [11]. Despite the increase in running time, these results, to some degree, benchmark the quality of the solutions found in Ceschia and Schaerf [11].

An outline of this paper is as follows. In Section 2 we introduce some required terminology

and provide a more formal definition of the problem at hand. Section 3 gives an overview of the proposed ALNS procedure, while Section 4 details our computational study and, in particular, the comparison with Ceschia and Schaerf [11]. Finally, the main conclusions from this study are drawn in Section 5.

2. Problem definition

In this section we formalize the definition of the DPAS problem considered. This follows the original problem description given in Ceschia and Schaerf [11]. To avoid confusion all terminology from Ceschia and Schaerf [11] is retained. Furthermore, we refer the reader to the appendix for a list of the notation used throughout the paper. In any instance of the DPAS problem it is assumed that there is a set of patients $\mathcal{P} = \{1, 2, \ldots, P\}$ requiring admission to a hospital over a pre-specified time horizon of known length. This patient set is partitioned into a set of male patients, $\mathcal{M}$, and a set of female patients, $\mathcal{F}$. The planning horizon is divided into a set of consecutive days, denoted by the set $\mathcal{D} = \{1, 2, \ldots, D\}$, and indicates for how long the admission schedule is required. The problem is dynamic in the sense that not all patients are known beforehand and information is revealed gradually. It is possible to reschedule patients each day within the planning horizon when new information is available. Associated with each patient $p \in \mathcal{P}$ are several important dates, and these are known in advance. The first, $d_p^{reg} \in \mathcal{D}$, states the day on which patient p is registered in the hospital's administrative records. Although a patient's registration date is known in advance, knowledge of this cannot be used prior to this date.[1] The second, $d_p^{plan} \in \mathcal{D}$, indicates the day on which patient p is expected to be admitted to the hospital. We consider this date as the first possible day of admission. Finally, $d_p^{max} \in \mathcal{D}$, gives the last possible day patient p can be admitted where $d_p^{plan} \leq d_p^{max}$. Thus the patient p can be admitted into the hospital on any day between d_p^{plan} and d_p^{max}, both days included. One can think of d_p^{plan} as the preferable day to admit patient p, while d_p^{max} can be thought of as the latest date to which we can postpone patient p. Obviously, by setting $d_p^{plan} = d_p^{max} = d_p^{reg}$, the DPAS problem can incorporate emergency patients as these patients cannot be postponed. Additionally, a patient's age, required treatment(s), length of stay, l_p (given in number of nights)[2], and whether the patient possesses a so-called *overstay risk* are estimated beforehand. The latter indicates that there is the risk that the patient will stay *one* day longer. If a patient has an overstay risk, then it is desirable for their assigned room to have an empty bed on their planned discharge date. The problem takes it's offset in Ceschia and Schaerf [11] and does not consider overstays of arbitrary length. Finally, in order to be consistent, we assume that $D \geq \max_{p \in \mathcal{P}}\{d_p^{max} + l_p + 1\}$.

A hospital is assumed to consist of several departments, where each department itself further consists of several rooms. Departments typically specialize in the treatment of one or more illnesses. We denote the set of all rooms available at the hospital as $\mathcal{R} = \{1, 2, \ldots, R\}$. Each room $r \in \mathcal{R}$ has a certain capacity κ_r; this indicates the number of beds in the room. Room capacity is typically one, two, or four beds. Each room is assumed to be equipped with a set of features.

[1] The registration dates are given in the available data instances. However, in a real-life situation these dates will only be available when the patient arrives in the system.

[2] The length of stay is the number of nights the patient needs the bed. Only in-patients are considered as they need a bed for at least a single night.

Features indicate the presence of specific equipment or properties, which can be necessary (or only preferable) for patients. Finally, each room is also assumed to have a gender policy. There are four possible gender policies, and these are listed in Table 1. The Same Gender (SG) policy is the most used policy in many hospitals. An illustration of a patient admission schedule is given in Figure 1. For this fictitious hospital, the rooms in Department 1 have the SG, Department 2 has the All (A) policy, Department 3 has the Male (Ma) policy, and Department 4 has the Female (Fe) policy. Note that the gender policy does not necessarily have to be the same for all rooms of a department. Rooms with the SG policy are solution wise of interest and we let $\mathcal{R}^{SG}$ be the subset of rooms having the SG policy.

Policy	**Description**
SG	Both genders are allowed, but not at the same time.
A	There is no gender restriction.
Ma	Only male patients are allowed.
Fe	Only female patients are allowed.

Table 1: The available gender policies

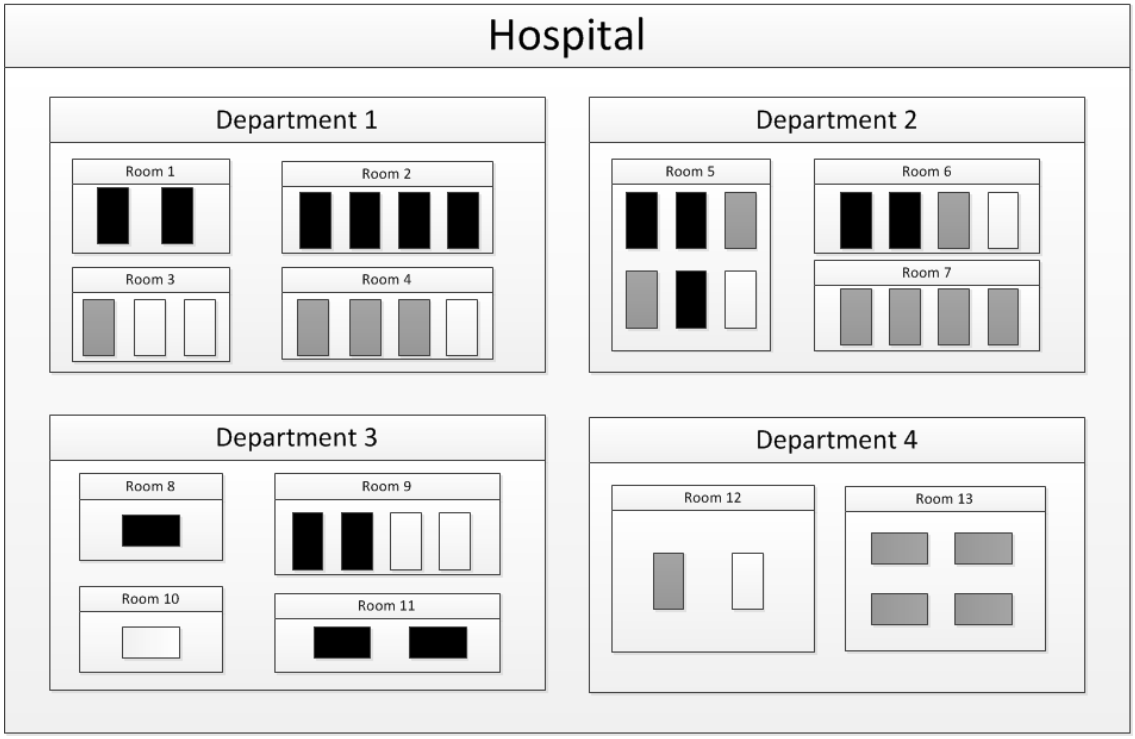

Figure 1: An example schedule for a hospital. The black beds are assigned to male patients, while gray beds are assigned to female patients. White beds are empty

When finding a solution to the DPAS problem a variety of constraints, some hard and some soft, exist. Hard constraints must be respected and concern things like room capacity, patient age, department specialism, and required room features. The capacity of a room is stated in terms of

beds and indicates how many patients can be assigned to the room on a given day. Furthermore, a patient can only be treated in a department if the age of the patient satisfies the age limit of the department. This is a crude attempt to model things such as a pediatric department. Similarly, a patient must be treated in a department which has the specialism the patient requires, and cannot be assigned to a room not equipped with properties the patient needs.

Soft constraints, on the other hand, are used to model preferences or desirable properties of the solution. Ideally, they should be enforced, but it is not necessary; they can be violated at a cost. These include things such as: department main specialism, preferred room feature, transfers, gender, room size, admission delay, and overstay risk. To elaborate, ideally a patient should be assigned a department whose main specialism is in the treatment the patient needs; preferred properties for a patient should be in the room they receive; the transferring of a patient between rooms should be avoided; the patients assigned to a room should be consistent with the room's gender policy; a patient should receive a room with capacity in line with their preferences; and if any of the patients assigned to a room have an overstay risk, the room should have an available bed on their expected discharge date (just in case). Table 2 summarizes the different constraints of the problem, their type and, if it is a soft constraint, the penalty incurred when violating the given constraint once. The values of the weights are taken directly from Ceschia and Schaerf [11]. This is to allow an accurate comparison of the solutions obtained. The highest weight is given to transferring the patient. This is in line with common practice, where transfers are really only used as a last option to maintain or gain feasibility. The two last constraints, concerning delaying the admission and the overstay risk, are extensions of the static version of the problem. The former penalizes delaying a patient's admission to the hospital; for the admission date this can, however, not be higher than d_p^{max}. The latter concerns the overstay risk. This tries to ensure that patients, who have the potential to stay longer than expected, are assigned to a room that has a surplus capacity on their discharge dates. If such patients do stay longer than is planned, and an attempt has not been made to reserve capacity, so-called *overcrowding* may occur.

Constraints	**Type**	**Weight**
Room capacity	hard	-
Patient age (PA)	hard	-
Department specialism (DS1)	hard	-
Department main specialism (DS2)	soft	20
Needed room feature (RF1)	hard	-
Preferred room feature (RF2)	soft	20
Transfer (Tr)	soft	100
Gender (Gr)	soft	50
Room size preference (RS)	soft	10
Admission delay (De)	soft	2
Overcrowd risk (OvR)	soft	1

Table 2: Constraints and their weights[3]

Given the constraints defined above, the objective of the DPAS problem is to find an assign-

[3]Values taken directly from Ceschia and Schaerf [11].

ment of patients to rooms that minimizes the sum of the penalties incurred from soft constraint violations. Violating hard constraints is not permitted. As such, any (patient, room) combinations that would result in a hard constraint violation can obviously be removed from consideration. For example, if a patient needing a certain treatment is placed in a room which is in a department that does not specialize in the treatment of that specific illness, this particular (patient, room) combination can be eliminated.

3. Solution approach

The size and complexity of the DPAS problem makes the application of exact mathematical programming procedures difficult. A few ILP based exact procedures have been proposed in the literature for both the static and dynamic variants of the PAS problem, and all have concluded that only small instances of the problem are tractable with such approaches, see e.g. Ceschia and Schaerf [7, 11], Range et al. [9]. Metaheuristic approaches are therefore an attractive alternative as they can overcome such tractability issues. Being heuristic in nature they are, however, unable to independently provide a certificate of quality on the obtained solution. Nonetheless, they make it possible to consider large problem instances and, if designed well, can produce good solutions. In this section we present and describe such an approach for solving the DPAS problem. The developed methodology repeatedly applies a SA-based ALNS algorithm throughout the planning horizon. To aid the exposition, this section has been divided into four parts. To have a concise description of the problem we begin with a mathematical model for the static version in Section 3.1. In Section 3.2 we continue with a short discussion on the dynamic nature of the problem. Section 3.3 provides a general description of a SA-based ALNS algorithm, while Section 3.4 elaborates on the application of the technique to the DPAS problem.

3.1. A mathematical model for the static PAS

We introduce a mathematical formulation of a static version of the DPAS problem, not to be able to solve it as such, but rather as a means to state the problem precisely. The mathematical formulation presented is a static model including all patients in the solution, and it does not exclude patients who have not been registered yet. Hence, in this static model we have all future information available. It is, however, possible to apply this to the dynamic setting where some variables are then fixed. We describe this application of the model in Section 3.2.

Each patient $p \in \mathcal{P}$ has a window of dates in which (s)he can be admitted. We denote this window as $\mathcal{D}_p = \{d_p^{plan}, \ldots, d_p^{\max}\} \subseteq \mathcal{D}$. A patient $p \in \mathcal{P}$ can be allocated to a room $r \in \mathcal{R}$ if the hard constraints PA, DS1, and RF1 are satisfied for the patient. We let $\mathcal{R}_p$ be the set of feasible rooms for patient $p \in \mathcal{P}$. For each patient $p \in \mathcal{P}$ and room $r \in \mathcal{R}$ we denote the daily cost of admitting the patient into the room as c_{pr}. This cost corresponds to the sum of the soft constraint penalties DS2, RF2, RS, and Gr (for the policies Ma and Fe) from Table 2, if any of these are violated when assigning patient $p \in \mathcal{P}$ to room $r \in \mathcal{R}_p$. If $r \notin \mathcal{R}_p$, then we set $c_{pr} = \infty$. This cost is independent of the day on which the patient occupies the room. For gender-segregated rooms, i.e. rooms following the SG policy, we denote c^G the daily cost of admitting patients of both genders into the room. We let this cost be equal to the penalty Gr given in Table 2. We use c^D to denote the cost per day of delaying a patient. This corresponds to the penalty incurred

when violating soft constraint De from Table 2. The parameter $O_p \in \{0,1\}$ is used to indicate whether or not patient $p \in \mathcal{P}$ has a risk of staying an additional day, where $O_p = 1$ if and only if patient p has a risk of staying an additional day, while c^O denotes the unit overcrowd cost for having potentially more patients in the room than the room's capacity. This unit overcrowd penalty is associated with violation of soft constraint OvR in Table 2. Finally, c^T denotes the cost of a transfer and corresponds to violation of soft constraint Tr from Table 2.

Binary decision variables $x_{prd} \in \{0,1\}$ are used to indicate whether or not patient $p \in \mathcal{P}$ occupies a bed in room $r \in \mathcal{R}$ on the night following day $d \in \mathcal{D}$, where $x_{prd} = 1$ if the patient p occupies a bed in room r on the night following day d and $x_{prd} = 0$ if not. An auxiliary set of variables $\overline{x}_{prd} \in \{0,1\}$ indicates that the patient $p \in \mathcal{P}$ may occupy a bed in room $r \in \mathcal{R}$ on the night following day $d \in \mathcal{D}$ after discharge due to the overstay risk, and this set is used to calculate the potential overcrowding of a room. That is, if the patient p has an overstay risk $O_p = 1$ then the patient may stay an extra night in the room in which (s)he resides, thus $\overline{x}_{prd} = 1$ if patient patient p is admitted on day $d - l_p$ and resides in the room r on day d. In addition, binary variables $t_{pd} \in \{0,1\}$ are equal to one if patient $p \in \mathcal{P}$ is transferred from one room to another, from day $d-1 \in \mathcal{D}$ to day $d \in \mathcal{D}$, and zero otherwise. For consistency we define $t_{p1} = 0$ for all $p \in \mathcal{P}$. The day of admission for patient $p \in \mathcal{P}$ is controlled by the binary variable $y_{pd} \in \{0,1\}$; this equals one if and only if patient $p \in \mathcal{P}$ has day $d \in \mathcal{D}$ as their day of admission. For each room and each day we let the binary variables $f_{rd}, m_{rd} \in \{0,1\}$ indicate whether or not at least one female or at least one male is present in the room, respectively, i.e. $f_{rd} = 0$ only if no female is present in room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$ and $m_{rd} = 0$ only if no male is present in room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$. The binary variable $b_{rd} \in \{0,1\}$ indicates that both genders are present in the room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$, where $b_{rd} = 0$ only if either no females are present or no males are present in the room. Finally, $z_{rd} \geq 0$ measures the potential number of patients beyond the capacity of the room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$.

Some of the variables can be fixed to zero. As it is only possible to admit a patient $p \in \mathcal{P}$ during the days in $\mathcal{D}_p$, $y_{pd} = 0$ for all $d \notin \mathcal{D}_p$. Likewise, $x_{prd} = 0$ if $r \notin \mathcal{R}_p$ or if the day d is either before d_p^{plan} or after $d_p^{\max} + l_p - 1$.

For a solution to be feasible it has to satisfy the set of hard constraints. These are listed below. First of all, the rooms can only accommodate as many patients as the room has beds for:

$$\sum_{p \in \mathcal{P}} x_{prd} \leq \kappa_r, \quad r \in \mathcal{R}, d \in \mathcal{D}. \tag{1}$$

This set of constraints measures the occupation of the room by patients within the length of stay for the patient, but does not take the potential occupation of patients having an overstay risk into account. The potential overcrowding of a room due to the overstay risk is calculated in constraints (9). Next, the individual patients have to be admitted on precisely one day within the planning horizon:[4]

$$\sum_{d \in \mathcal{D}_p} y_{pd} = 1, \quad p \in \mathcal{P}. \tag{2}$$

[4]In real-life problems wards may be working close to capacity with regards to beds and, as a consequence, it may not be possible to admit the patient at all. It is possible to penalize this by adding a variable playing the role of an explicit artificial variable making it unattractive to not being able to admit the patient.

If patient p is admitted on day $\overline{d}$, the patient must appear in a room the following $l_p - 1$ nights

$$\sum_{r \in \mathcal{R}_p} x_{prd} \geq y_{p\overline{d}}, \quad p \in \mathcal{P}, \overline{d} \in \mathcal{D}_p, d = \overline{d}, \ldots, \overline{d} + l_p - 1. \tag{3}$$

Constraints (1)-(3) constitute the core of the model. The remaining constraints serve different purposes. These include indicating the presence of male and female patients in rooms (to calculate the violation of the gender constraints in rooms with the SG policy), identifying when transfers happen, and calculating the potential overcrowding of rooms.

The first set of constraints is used to segregate genders in rooms having the SG gender policy. The presence of a female patient is determined by

$$f_{rd} \geq x_{prd}, \quad p \in \mathcal{F}, r \in \mathcal{R}_p \cap \mathcal{R}^{SG}, d \in \mathcal{D}_p, \tag{4}$$

while the presence of a male patient is determined by

$$m_{rd} \geq x_{prd}, \quad p \in \mathcal{M}, r \in \mathcal{R}_p \cap \mathcal{R}^{SG}, d \in \mathcal{D}_p. \tag{5}$$

This allows us to calculate when both genders are present using the following constraints:

$$b_{rd} \geq f_{rd} + m_{rd} - 1, \quad r \in \mathcal{R}^{SG}, d \in \mathcal{D}. \tag{6}$$

Thus, b_{rd} is calculated for all same-gender policy rooms on all days. The transfer of patients is enforced using the following constraints:

$$t_{pd} \geq x_{prd} - x_{prd-1} - y_{pd}, \quad p \in \mathcal{P}, r \in \mathcal{R}_p, d = 2, \ldots, D. \tag{7}$$

In other words, a transfer is recorded if patient $p \in \mathcal{P}$ does not stay in the same $r \in \mathcal{R}$ on consecutive stays during their stay. Without the y_{pd} term the constraint would also record a transfer when admitting patient p to the hospital (since the patient would not have been assigned the same room on the previous day).

If a patient p has an overstay risk, i.e $O_p = 1$, then the patient potentially occupies a bed for an extra day in the last room of their planned stay. This is modeled as follows:

$$y_{pd} + O_p x_{prd+l_p-1} \leq 1 + \overline{x}_{prd+l_p}, \quad p \in \mathcal{P}, r \in \mathcal{R}_p, d \in \mathcal{D}_p. \tag{8}$$

Having settled the potential occupation of a bed in room r the day after the planned discharge, the potential overcrowding of any room can be recorded with the following constraints:

$$\sum_{p \in \mathcal{P}} (x_{prd} + \overline{x}_{prd}) \leq \kappa_r + z_{rd}, \quad r \in \mathcal{R}, d \in \mathcal{D}. \tag{9}$$

The domain of each of the decision variables is given below.

$$y_{pd}, t_{pd} \in \{0,1\}, \quad p \in \mathcal{P}, d \in \mathcal{D} \tag{10}$$

$$x_{prd}, \overline{x}_{prd} \in \{0,1\}, \quad p \in \mathcal{P}, r \in \mathcal{R}, d \in \mathcal{D} \tag{11}$$

$$f_{rd}, m_{rd}, b_{rd} \in \{0,1\}, \quad r \in \mathcal{R}, d \in \mathcal{D} \tag{12}$$

$$z_{rd} \geq 0, \quad r \in \mathcal{R}, d \in \mathcal{D} \tag{13}$$

The objective consists of five terms. The first measures the time-independent penalty of assigning a patient to a room for one night. The second term calculates the cost of all transfers of patients, while the third term determines the penalty given for gender violations of rooms having the same-gender policy. The fourth term penalizes the potential overcrowding, and finally the fifth term penalizes the admission delay of patients. The objective can be stated as:

$$\begin{aligned} \text{minimize} \quad & \sum_{p \in \mathcal{P}} \sum_{d \in \mathcal{D}_p} \sum_{r \in \mathcal{R}_p} c_{pr} x_{prd} + c^T \sum_{p \in \mathcal{P}} \sum_{d \in \mathcal{D}} t_{pd} + c^G \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}^{SG}} b_{rd} \\ & + c^O \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} z_{rd} + c^D \sum_{p \in \mathcal{P}} \left(\sum_{d \in \mathcal{D}_p} dy_{pd} - d_p^{plan} \right) \end{aligned} \tag{14}$$

The overall aim of the problem is to minimize the penalties incurred from violations of soft constraints, while satisfying the constraints (1)-(9) and the domain of the variables (10)-(13).

The model presented above corresponds to the problem verbally stated and solved by Ceschia and Schaerf [11]. Ceschia and Schaerf [11] also present a mathematical formulation, but only for a simplified version of the problem, where it is not possible to delay patients' admission, nor is it possible to transfer patients. This greatly simplifies their mathematical formulation. In addition their mathematical formulation only checks whether or not there is a risk of overcrowding in a room, which is in contrast to the model they actually solve, where the amount of potential overcrowding is penalized. The authors clearly state that the mathematical formulation is a simplified version of a more elaborate formulation, and briefly devise directions for setting up a full mathematical formulation. The mathematical formulation we present should be viewed as such a full mathematical formulation of the problem.

In practice this model suffers from severe degeneracy and standard integer linear programming solvers are typically not able to solve the problem in reasonable time – see e.g. Ceschia and Schaerf [7]. Recall that we do not intend to solve this formulation directly by mixed integer linear programming, but instead we use a heuristic approach.

3.2. The dynamic setup

The DPAS problem is a dynamic problem and as such it calls for continual optimization based on the available information. We assume that the information on registered patients is made available once each day, after which an optimization problem is solved to identify which rooms patients will be assigned to in the future. Patients who are already admitted can only stay in the room from the day before or be moved to other rooms, resulting in a transfer, and naturally they cannot be postponed. Other registered patients can freely be moved by the optimization

approach. Formally, on each day $d \in \mathcal{D}$ we must optimize the assignment of the set of patients

$$\mathcal{P}_d = \{p \in \mathcal{P} | d_p^{reg} \leq d, p \text{ not discharged}\}$$

to rooms. We refer to $\mathcal{P}_d$ as the set of *active* patients on day d. The number of active patients on day $d \in \mathcal{D}$ is denoted by $A_d := |\mathcal{P}_d|$. As each patient $p \in \mathcal{P}$ has a last possible day of admission, d_p^{max}, the planning horizon for the optimization on day d is given by $\max\{d_p^{\max} | p \in \mathcal{P}_d\}$.

Suppose that we have solved all the days until day $d-1$ and want to solve the problem for day d. In terms of the model from Section 3.1 we exclude all patients (and corresponding variables) from $\mathcal{P} \setminus \mathcal{P}_d$, as they have been discharged or have not been registered yet. We obtain the solution from the day before and fix all patients $p \in \mathcal{P}_{d-1} \cap \mathcal{P}_d$ with $y_{pd'} = 1$ for a day $d' \leq d-1$ to be admitted on day d'. For these patients we also fix the rooms in which they have stayed up until day $d-1$. For patients who are not fixed we state that their earliest day of admission will be $\max\{d, d_p^{plan}\}$. Fixing these variables corresponds to setting up boundary conditions for the optimization of day d which depends on the irreversible decisions made on previous days.

When optimizing the assignment of patients $\mathcal{P}_d$ on a given day $d \in \mathcal{D}$, we must first construct an initial solution where newly registered patients, i.e. patients having $d_p^{reg} = d$, are included. To do this we use a simple greedy insertion algorithm. For the first day there are no previous daily assignments we must respect; however, for all subsequent days, we must respect the previous day's final patient to room assignment for the already admitted patients when constructing the solution. This greedy insertion heuristic is explained in more details in Section 3.4 as it is used extensively in the metaheuristic framework.

It is important to note that this constructive heuristic does not guarantee feasibility; a patient's admission date cannot be delayed indefinitely if infeasibility persists. The admission date of patient $p \in \mathcal{P}$ can only be delayed up until, and including, $d_p^{\max}$. When this does occur, the search for a compatible room for the patient is postponed until the algorithm has assigned all other patients. The admission date of the patient is then set to a date which exceeds the current day and for which a compatible room can be found. This results in an infeasible solution, violating either the requirement that a patient be admitted before his/her maximal admission date or the requirement that there is a fixed number of days in the planning horizon. To be comparable to Ceschia and Schaerf [12], these infeasibilities receive an extra penalty of 200 per occurrence to ensure that they are quickly remedied during the optimization process.

3.3. Simulated annealing based adaptive large neighborhood search

The main principle of local search is to iteratively move from one feasible solution x to a neighboring feasible solution x'. The set of neighboring solutions to a given solution is termed a *neighborhood* and each neighboring solution is obtained from the given solution by making a set of local changes. If the neighboring feasible solution has a better objective value, it is accepted and its neighbors are then considered, otherwise a different neighbor of the given solution is generated. The downside with this approach is that a solution that is only locally optimal is ultimately reached, and this halts the exploration of better solutions. Metaheuristics attempt to overcome local optima by allowing the search to move to worse solutions in an attempt to introduce some diversification. One well known approach for this is the SA approach described

originally in Metropolis et al. [13], but first adopted for optimization by Kirkpatrick et al. [14]. In a SA metaheuristic better solutions are always accepted; however, worse solutions are also accepted with a certain probability. This acceptance probability is continually decreased during the execution of the algorithm, ensuring that worse solutions are less likely to be accepted late in the process.

The process of decreasing the acceptance probability is often described as lowering the temperature of an annealing process (hence the name). We start out with an initial temperature $T := T_{start}$ and then decrease this temperature to $T := \alpha \cdot T$ at the end of each iteration, where $\alpha \in (0, 1)$. The acceptance probability of a new but worse solution x' is then calculated as

$$\rho_{acc} := \exp\left(-\frac{f(x') - f(x)}{T}\right)$$

The algorithm terminates when the temperature reaches a lower temperature T_{end}. For a more detailed description on the practical implementation of SA refer the reader to Aarts et al. [15].

Instead of making minor changes in moving from one solution to the next, it is possible to make more dramatic changes by destroying a significant part of the solution and then repairing it again by applying a constructive heuristic. This is the fundamental idea of so-called *destroy and repair* methods and large neighborhood search, see Shaw [16]. Destroying a part of the solution and then repairing it again tremendously enlarges a solution's neighborhood. This makes a full search of all neighboring solutions impractical, and consequently a SA framework is applied.

Pisinger and Ropke [17] observed that not all destroy and repair methods remain equally good throughout the course of a SA algorithm. In other words, some tend to improve the solution more in the beginning of the solution process, while others are better suited towards the end. The authors therefore suggest to assign modifiable weights to each of the destroy methods as well as weights to each of the repair methods. These weights are then continually updated over the course of the algorithm and are decided using an exponential smoothing of the performance of the individual methods. This gives rise to the so-called ALNS metaheuristic, which *adapts* its selection of the destroy and repair methods based on the respective performances of each of them.

The set of all such destroy and repair methods is given as $H = I \cup J$, where I is the set of repair methods and J is the set of destroy methods. For each method $h \in H$ a weight w_h is defined and is based on the performance of the method; the better it performs the higher its weight. Well-performing methods should have a higher probability of being chosen than worse-performing methods. The probability of choosing a repair method $h \in I$ is given as

$$\pi_h := \frac{w_h}{\sum_{i \in I} w_i}, \quad h \in I,$$

while the probability of choosing a destroy method $h \in J$ is denoted

$$\pi_h := \frac{w_h}{\sum_{j \in J} w_j}, \quad h \in J.$$

The performance of the methods is based on observing the changes in objective value over a given number of iterations, n_{iter}. For each method, s_h defines the accumulated score for method $h \in H$, and v_h states the number of times a method has been used. For an iteration where method h is

chosen to move from solution x to solution x' the following is added to s_h (assuming a minimization problem)

$$\sigma := \begin{cases} \sigma_1, & \text{if } f(x') < f(x^*) \\ \sigma_2, & \text{if } f(x^*) \leq f(x') < f(x) \\ \sigma_3, & \text{if } f(x) \leq f(x') \text{ and } x' \text{ accepted} \\ 0, & \text{otherwise} \end{cases}$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$. Note that $f(x)$ gives the objective value of solution x, and x^* is the best found solution so far. In the case where a new best solution is found σ_1 is added, while in the case where an improvement over the current solution is found, σ_2 is added. If the neighboring solution is not an improvement but is still accepted, then σ_3 is added to the score.

After n_{iter} iterations the weights are updated as follows

$$w_h := \begin{cases} (1-\gamma) \cdot w_h + \gamma \cdot \dfrac{s_h}{v_h}, & \text{if } v_h > 0 \\ w_h, & \text{if } v_h = 0 \end{cases}$$

after which s_h and v_h are reset to zero.The reaction factor, $\gamma \in [0, 1]$, indicates how much emphasis is placed on the score of the most recent n_{iter} iterations. If $\gamma = 0$, then the weights are kept constant at their initial levels, while $\gamma = 1$ only uses the score of the most recent n_{iter} iterations. The update procedure for w_h resembles the reward function often found in reinforcement learning (see e.g. Sutton [18]), a well known machine learning technique, which utilizes a trial-and-error search and a delayed reward. Here, each method is rewarded based on its ability to find a good solution. Using adaptive weights for the methods, the algorithm self determines the likelihood with which each destroy and repair method is chosen at a given iteration. A similar procedure is described in [19] for operator selection within a metaheuristic framework.

3.4. Application of ALNS to the DPAS problem

In what follows we describe an application of the ALNS methodology to the DPAS problem. We present a framework consisting of three destroy methods and two repair methods, and describe each of these in detail. Note that when we remove a patient from a solution, the patient is removed for all days it occupies the assigned room from the current day and into the future. Consequently, when repairing the solution, by inserting the patient again, the patient will be inserted into a room for the remaining part of his/her admission. The proposed ALNS algorithm is run for each day $d \in \mathcal{D}$ of the planning horizon. We solve each day in turn, considering only the patients in the active set $\mathcal{P}_d$.

Given a solution to a particular day, the destroy methods focus on identifying a number of patients, $N \in \mathbb{Z}_0^+$, to be removed from the solution and subsequently reinserted. If N is small (say close to one or two), then the method will resemble a standard local search approach. If, on the other hand, N is large (close to $|\mathcal{P}|$), then the resulting solution could become quite different, essentially running a reinsertion heuristic for all the patients. We choose the value of N to be within $4 \leq N \leq \min\{100, \xi \cdot A_d\}$ using a uniform distribution, where $\xi \in]0; 1]$ and A_d is the number of active patients in the problem. The value 100 is chosen to limit the computational time used. The parameter ξ depends on the instance size.

The first destroy method we use is termed `Random Removal`. As the name suggests, this method

randomly selects N patients using a uniform distribution and removes them from the solution, the aim being to simply randomize the selection in order to explore other parts of the solution space.

The second removal heuristic aims at removing patients with the highest cost in the solution, and is termed `Worst Removal`. Such patients have obviously not received their most favorable room, and we are therefore interested in finding them another room. To ensure that we do not always select the worst positioned patients, we apply the following randomization which favors the selection of the worst cost patients. Let $\overline{p}_1, \ldots, \overline{p}_{A_d}$ be a list of patients sorted in decreasing order of patient cost, which corresponds to the cost of soft constraint violation directly related to the patient, i.e. the penalty for the patient being placed in a given room, the penalty for violation of the gender policy, the penalty from the admission delay, and the penalty for not having a bed when there is an overcrowd risk. Suppose that r is uniformly distributed on the interval $[0, 1[$, and that $\beta \geq 1$ is the randomization factor, we determine $1 \leq n \leq A_d$ to be

$$n := \lfloor r^{\beta} \cdot A_d \rfloor + 1,$$

and select patient $\overline{p}_n$ for removal. After removing patient $\overline{p}_n$ we decrease the size of A_d by one and reset the indices of $\overline{p}_1, \ldots, \overline{p}_{A_d}$ accordingly. This is repeated until N patients are removed. We use the above formula to set n in order to have some flexibility in the number of patients to remove. For example, if $\beta = 1$, this selection corresponds to the random removal; however, when we increase β, the likelihood of selecting patients with lower indices increases. If, in the extreme case, $\beta \to \infty$, we would always select the first element of $\mathcal{P}_d$. We thereby have a compromise between always selecting the worst patient and selecting a patient at random.

As patients have different characteristics, removing two patients at random does not necessarily mean that their respective room assignments can be feasibly swapped. Patient differences can arise in admission timing or in treatment requirements. Consequently, the third removal heuristic – inspired by Shaw [16] – is based on trying to remove *related* patients. This is termed `Related Removal`. A patient is randomly selected and the remaining $N-1$ patients are sorted in decreasing order of relatedness. We define relatedness as a weighted sum of four factors:

1. $\texttt{rel}_1(p_1, p_2)$: The number of days the stays of patients p_1 and p_2 overlap divided by the number of days the patient with shorter stay has.
2. $\texttt{rel}_2(p_1, p_2)$: The proportion of all rooms that are feasible for patients p_1 and p_2.
3. $\texttt{rel}_3(p_1, p_2)$: This is equal to one if patients p_1 and p_2 have the same gender and is zero otherwise.
4. $\texttt{rel}_4(p_1, p_2)$: A room is related cost-wise for the patients if the room has the exact same cost for these, i.e. $c_{p_1 r} = c_{p_2 r}$. The relatedness based on a given room is then the cost of the lowest cost patient divided by the cost of the highest cost patient i.e. $\min\{c_{p_1 r}, c_{p_2 r}\} / \max\{c_{p_1 r}, c_{p_2 r}\}$, where we use the convention that the relatedness is 1 if $\max\{c_{p_1 r}, c_{p_2 r}\} = 0$. Thus the average room-based relatedness over feasible rooms is considered a relatedness measure. This is expressed mathematically as:

$$\texttt{rel}_4(p_1, p_2) := \frac{1}{|\mathcal{R}_{p_1} \cap \mathcal{R}_{p_2}|} \sum_{r \in \mathcal{R}_{p_1} \cap \mathcal{R}_{p_2}} \frac{\min\{c_{p_1 r}, c_{p_2 r}\}}{\max\{c_{p_1 r}, c_{p_2 r}\}}.$$

Each of these measures is normalized and has a value on the interval $[0, 1]$. Their relative im-

portance does, however, differ. For example, two patients who are not admitted in overlapping periods are not related, nor are two patients who have no feasible rooms in common. Our final measure of relatedness is hence

$$\texttt{rel}(p_1, p_2) := \begin{cases} 0, & \text{if } \texttt{rel}_1(p_1, p_2) = 0 \text{ or } \texttt{rel}_2(p_1, p_2) = 0 \\ \sum_{i=0}^{4} \varrho_i \cdot \texttt{rel}_i(p_1, p_2), & \text{otherwise} \end{cases}$$

where ϱ_i reflects the relative importance of relatedness measure i.

After the solution has been destroyed through the removal of N patients, the removed patients must be reinserted into the resulting partial solution. The first repair method is very simple and is termed `Greedy Insertion`. In this method patients are greedily inserted based on the cheapest insertion. The second, termed `Regret Insertion`, attempts to minimize the cost that will be incurred from not yet admitted patients when inserting a given patient. Recall that it is possible to postpone the admission of a not yet admitted patient p by up to $d_p^{\max} - d_p^{plan}$ days.

Given a partial solution in which N of the A_d active patients are not yet inserted, for each of the removed patients a set of triples can be generated. Each triple is of the form (p, r, k) and states an insertion possibility for patient p. In particular, it states that patient p is assigned room r with k days of delay from the d_p^{plan}. Associated with each such triple is an assignment cost, c_{prk}, reflecting the increase in cost that would be incurred if the corresponding insertion were made. If the assignment indicated by a given triple is not feasible, then its cost is assumed to be infinitely large. The `Greedy Insertion` sorts all such triples in increasing order of cost and iteratively considers them. At any given insertion, the triple resulting in the smallest immediate increase in cost is considered. Due to the overstay risk and gender constraints, the cost of all triples needs to be updated after an insertion has been made. This process is repeated until either no more patients can be inserted or all patients have been feasibly inserted.

Greedily inserting patients has the potential to spoil the possibility of placing other patients later in the insertion process. The `Regret Insertion` is an attempt to combat this. In other words, this method iteratively inserts the patient that leaves the greatest number of possibilities for the remaining patients. Ties are broken arbitrarily. Essentially, the `Regret Insertion` method focuses on patients that we would regret the most if not inserted immediately. We define $\overline{c}_{pr} := \min_k\{c_{prk}\}$ to be the cheapest day to insert patient p in room r. If we then sort the $\overline{c}_{pr}$ values (there is at most one for each room) in increasing order and let $r(j)$ be the room at position j in the sorted list, we obtain the following general regret cost

$$v_p^k := \sum_{j=2}^{k} \left(\overline{c}_{pr(j)} - \overline{c}_{pr(1)}\right).$$

This is an indication of how much worse it will be to insert patient p in his/her k'best room compared to his/her best room. Selecting the patient p to insert with the largest value of v_p^k will try to minimize the regret by reducing the ill-effect caused by not being able to place patients in good rooms towards the end of the algorithm.

Algorithm 1 provides an overview of the full SA based ALNS metaheuristic. For each temperature, we execute a certain number of trials (*ntrials* in Algorithm 1). Each trial itself consists

Algorithm 1 Adaptive Large Neighborhood Search with Simulated Annealing

1: **Input:** T_{start}, T_{end}, n, ξ, α, d
2: $x_0 \leftarrow$ `runGreedyConstruction()`
3: $x \leftarrow x_0$
4: $x_{best} \leftarrow x_0$
5: $T \leftarrow T_{start}$
6: $w \leftarrow$ `initializeMethodWeights()`
7: **while** $T > T_{end}$ **do**
8: **for** $i \leftarrow 1$, $ntrials$ **do**
9: $s \leftarrow$ `resetMethodScores`()
10: **for** $j \leftarrow 1, 100$ **do**
11: $N \leftarrow$ `random`([4,`min`$(100, \xi \cdot A_d)$])
12: `selectDestroyMethod`(w)
13: $\mathcal{P}' \leftarrow$ `remove`(x, N)
14: `selectRepairMethod`(w)
15: $x' \leftarrow$ `repair`$(x, \mathcal{P}')$
16: $\delta \leftarrow f(x') - f(x)$
17: $\rho_{acc} \leftarrow \exp\left(\frac{-\delta}{T}\right)$
18: $r \leftarrow$ Random([0,1])
19: **if** $\delta < 0$ or $r < \rho_{acc}$ **then**
20: $x \leftarrow x'$
21: **if** $f(x) < f(x_{best})$ **then**
22: $x_{best} \leftarrow x'$
23: **end if**
24: $s \leftarrow$ `updateMethodScores`()
25: **end if**
26: **end for**
27: $w \leftarrow$ `updateMethodWeights(s)`
28: **end for**
29: $T \leftarrow \alpha \cdot T$
30: **end while**
31: **return** x_{best}

of 100 iterations. In other words, we allow exactly 100 solutions to be considered on any trial. Again the value of 100 has been chosen to limit the run time of the algorithm. The value of $ntrials$, on the other hand, varies with problem size; we set it to $e \cdot R$, where e is a parameter we tune. The function `runGreedyConstruction()` (line 2) provides an initial solution using the `Greedy Insertion` repair method, while the functions `initializeMethodWeights()` (line 6), `resetMethodScores()` (line 9) are used to initialize the method weights, w_h, and the scores s_h for each destroy and repair method. Initially, the weights are chosen such that probability of choosing any destroy and repair method is equally likely. The functions `updateMethodScores()` (line 24) and `updateMethodWeights(score s)` (line 27) simply update the method weights and scores during the course of the algorithm. The former is updated after observing the changes in objective function value (see Section 3.3), while the latter is updated using the scores obtained after 100 iterations. The selection of the destroy and repair methods is controlled by the functions `selectDestroyMethod`(weights ω) (line 12) and `selectRepairMethod`(weights ω) (line 14), respectively. Each function takes the method weights as an argument. The actual removal and reinsertion of the patients occurs in functions `remove(`x`,`N`)` (line 13) and `repair(`x`,`$\mathcal{P}'$`)` (line 15). The set $\mathcal{P}'$ contains the patients which are first removed and subsequently reinserted.

4. Computational results

In this section we test and compare the performance of the proposed metaheuristic on the set of publicly available instances provided by Ceschia and Schaerf [11][5]. These artificially generated instances attempt to reflect reality and are divided into nine families of 50 instances. Each family is characterized by a size (`small`, `medium`, or `large`) and a horizon length (`short`, `mid`, or `long`). The size of the instance stipulates the number of departments, the number of rooms, the number of features, and the number of specialisms, while the horizon length indicates the number of days in the planning horizon. In particular, the `short` instances consider 14 days, the `mid` instances look at 28 days, and the `long` instances include 56 days. The number of patients in an instance depends on the horizon length; it doubles if the horizon length doubles. An overview of the instance families is provided in Table 3. Comparing the performance of our algorithm on the same data sets as those used in Ceschia and Schaerf [11] allows us to not only conclude something about the efficiency of the proposed algorithm, but also to benchmark the quality of the results obtained using the SA approach of Ceschia and Schaerf [11].

The ALNS framework is programmed in C++, and all experiments have been performed on a dedicated Intel(R) Xeon(R) CPU X5550 @ 2.67GHz with 24 gigabytes of main memory running Ubuntu Linux version 14.04. The framework has several parameters. We omit an extensive discussion on the parameter tuning of the algorithm here, and simply state that the best parameter values are obtained through a direct comparison of the average objective value (together with its standard deviation) for multiple runs of the algorithm over a range of initially specified values for each parameter. The chosen values for each parameter are the following: $T_{start} = 30$, $T_{end} = 0.5$, $e = 1.5$, and $\alpha = 0.5$. Depending on the instance size, the parameter ξ is equal to 0.3 (`small`), 0.2 (`medium`), or 0.075 (`large`). The reaction factor γ is set to 0.1, and the weights on the relatedness measures are $\varrho_1 = 4, \varrho_2 = 3, \varrho_3 = 2$, and $\varrho_4 = 1$, respectively. We reward routines that produce solutions that are accepted with scores of $\sigma_1 = 25, \sigma_2 = 15$, and $\sigma_3 = 5$, respectively. Finally, the randomness factor for worst removal, β, is set to 0.2.

Family	Instances	Departments	Rooms	Features	Patients	Specialisms	Days
`small-short`	50	4	8	4	50	3	14
`small-mid`	50	4	8	4	100	3	28
`small-long`	50	4	8	4	200	3	56
`medium-short`	50	6	40	5	250	10	14
`medium-mid`	50	6	40	5	500	10	28
`medium-long`	50	6	40	5	1000	10	56
`large-short`	50	8	160	6	1000	15	14
`large-mid`	50	8	160	6	2000	15	28
`large-long`	50	8	160	6	4000	15	56

Table 3: Instance overview

The results for each of the instance families are summarized in Table 4. Each instance of each

[5] Available at http://satt.diegm.uniud.it/projects/pasu/ and last accessed on August 18th, 2016.

family has been run 10 times, and each row of the table provides descriptive statistics for the 50 instances which comprise the family. Comparing to the results in Ceschia and Schaerf [11], for each family we report the largest improvement of any instance, the worst improvement of any instance, and the average improvement (all three given as a percentage). We also report the number of feasible solutions found by both approaches (CS refers to the approach in [11], while ALNS refers to the approach presented in this paper). For the ALNS approach we also report, in parentheses, the number of instances for which it produced a new best known solution. Finally, we also report the average time taken to solve each family size, as well as the average time taken to solve a day of the planning horizon (both in seconds). The latter is simply the average time divided by the number of days in the planning horizon. Note that in Ceschia and Schaerf [11] all tests were run on an Intel Core i7 @ 1.60 GHz. We estimate that this is a factor 1.8 times slower[6] than our processor. To account for this difference and allow an accurate comparison between the two algorithms we therefore also report adjusted times (Adj.) in Table 4.

	Improvement (%)			Solutions		Time (s)		
Family	Best	Worst	Avg.	CS	ALNS	Avg.	Per Day	Adj.
`small-short`	−16,54	5,11	−1,35	50	50 (27)	2,11	0,15	3,80
`small-mid`	−3,57	12,46	1,27	50	50 (13)	8,55	0,31	15,39
`small-long`	−4,81	7,02	1,74	49	50 (8)	39,52	0,71	71,14
`medium-short`	−6,99	3,37	−0,51	50	50 (25)	61,84	4,42	111,31
`medium-mid`	−5,37	2,72	−0,48	47	49 (31)	273,69	9,77	492,64
`medium-long`	−4,30	2,95	−1,25	49	50 (41)	1384,60	24,73	2492,28
`large-short`	−7,56	−0,24	−3,46	48	48 (48)	2252,28	160,90	4054,10
`large-mid`	−16,00	−0,55	−8,06	49	49 (49)	5346,35	190,94	9623,43
`large-long`	−21,95	−7,16	−13,90	48	49 (49)	22 407,49	400,13	40 333,48

Table 4: Summary of results on benchmark instances

Three noticeable trends can be observed from the results in Table 4. The first is that the ALNS approach appears to perform better the larger the instance size gets. For all instances, other than the `small` instances, the ALNS algorithm finds, on average, better quality solutions. In some cases the improvement is substantial. For example, on the `large-long` family we report objective values that are up to 22% better in some cases and on average 14% better. Furthermore, on the `large-long` instances we find new best known solutions for all instances. The second obvious trend is that the computation time drastically increases as the problem size grows. The solution times range from seconds on the `small-short` instances to hours on the `large-long` instances. A run time of 60 seconds per day of the planning horizon is permitted by Ceschia and Schaerf [11]. No artificial time limit is imposed in our algorithm. The results for the `small` and `medium` families adhere to this (even considering the adjusted run times); however, the same is not true for the `large` families. In the worst case, i.e. the `large-long`, the algorithm spends nearly seven minutes per day. It was a conscious decision not to enforce an artificial limit as we wanted to

[6] According to `http://www.cpubenchmark.net`, viewed on 21.06.2016, our processor has a benchmark of 5416, while that used in Ceschia and Schaerf [11] has a rating of 3049.

verify the quality of the solutions proposed in Ceschia and Schaerf [11]. It is not trivial how best to enforce the 60 second limit per day. Should all days have the same limit? Or should the days early in the time horizon be allocated more time? Naturally, the daily problems get easier to solve towards the end of the horizon as the number of registered patients decreases. Also, 60 seconds per day, irrespective of problem size, seems rather unrealistic. There are significantly more feasible solutions for the `large-long` instances than the `medium-long` instances. As such, a 60 second limit seems too general. Regardless of time limits, we have shown that solutions obtained on the `large-long` instances by Ceschia and Schaerf [11] can, in some cases, be significantly improved in terms of objective value. Finally, the third observation is that the ALNS approach is always at least as good at finding feasible solutions as the approach by Ceschia and Schaerf [11]. For example, on the `medium-mid` instances the ALNS approach finds a feasible solution on 49 of the 50 instances, while the approach by Ceschia and Schaerf [11] finds 47.

	ALNS*	CS*	ALNS (Avg.)	CS (Avg.)	Δ Best	Δ Avg.
`small-short`	2694,26	2723,02	2763,82	2761,07	−1,06	0,10
`small-mid`	6119,52	6058,18	6269,55	6165,21	1,01	1,69
`small-long`	12 394,14	12 179,73	12 627,70	12 383,17	1,76	1,97
`medium-short`	12 350,88	12 398,46	12 658,08	12 662,78	−0,38	−0,04
`medium-mid`	26 617,96	27 670,66	27 107,11	27 203,34	−0,53	−0,35
`medium-long`	61 816,90	62 616,82	62 530,64	63 483,94	−1,29	−1,43
`large-short`	38 931,65	40 302,35	39 589,10	41 099,36	−3,40	−3,67
`large-mid`	100 436,55	108 864,84	101 773,81	111 176,49	−6,51	−8,46
`large-long`	192 537,25	223 225,35	195 431,73	227 118,07	−13,75	−13,95

Table 5: Summary of best and average solutions on benchmark instances

Similar observations can be seen in Table 5. As a means to directly compare our metaheuristic (ALNS in the table) to that of Ceschia and Schaerf [11], we compute two different average objective values for both approaches across the 50 instances of each family. The first gives the average best objective value (*), while the second refers to the average of the mean objective values (Avg.). Recall that each instance is solved 10 times. The table also reports the percentage change between the best solutions and the average solutions for the two different methods (Δ Best and Δ Avg.). Table 5 shows that the best objective value is around 0.5% to 14% better, while the same is true for the mean objective values. For the `large` families the fact that the ALNS approach is better on average, compared to the best solution, indicates that the method consistently finds good solutions. For the `small` instances the solutions provided by the two approaches are comparable; however, the ALNS provides them in a fraction of the time. The apparent improvement on the `small-short` family is, however, somewhat misleading. This is due to the ALNS approach producing a significantly better solution on one of the instances (approximately 16.5% better).

Finally, all solutions have been validated using the publicly available solution validator provided by Ceschia and Schaerf [11][7]. This validator takes the solution for a given instance as input (i.e.

[7]Available online at `https://bitbucket.org/satt/pasu-instances/raw/master/validator.cc` and last accessed on August 18th, 2016.

the room assignment for each patient) and computes an objective value for the solution based on the definition of the problem by Ceschia and Schaerf [11]. Our results are therefore directly comparable to those of Ceschia and Schaerf [11] and our best found solution for each instance is available online at `http://www.ms.man.dtu.dk/Research/Instances`.

5. Conclusions

In this paper we have devised a metaheuristic for the DPAS problem. The method uses a SA framework and the ALNS procedure. This has resulted in an efficient and flexible method that can solve problems of different sizes and with different time horizons. For small problems solutions are found within 40 seconds, while for medium sized instances solutions are obtained within 25 minutes. Larger instances take somewhat longer, on average a few hours. However, it should be stressed that in reality the time taken to solve the problem on a given day of the planning horizon is more relevant since in a dynamic setting the problem only needs to be solved on a daily basis. For this, the times are much shorter, fractions of a second for the small instances and up to around seven minutes for the larger instances are needed. In most cases our method is superior to the method suggested by Ceschia and Schaerf [11]. For the large instances we report improvements in the range of 3-14% on average, but as large as 22% for individual instances. The taken to find these results is, however, significantly longer than the time limit which is granted to the method in Ceschia and Schaerf [11]. In the worst case, this is on average an increase by a factor of 12. Furthermore, the proposed approach finds solutions to more instances than Ceschia and Schaerf [11]. Out of the 450 instances used in the computational experiments the ALNS approach finds a feasible solution to 445 of them, five more than the method proposed in Ceschia and Schaerf [11].

If this work should be brought closer to a real-life implementation it is necessary to reduce running times for the large instances in order to get a tool that could work within the planning cycles seen at hospitals. Furthermore, ideas from the mathematical programming could be integrated into the heuristic framework in order to derive a matheuristic method.

References

[1] A. K. Hutzschenreuter, P. A. N. Bosman, I. Blonk-Altena, J. van Aarle, H. La Poutré, Agent-based patient admission scheduling in hospitals, in: M. Berger, B. Burg, S. Nishiyama(eds.) (Eds.), Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track, Estoril, Portugal, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 45–52, 2008.

[2] N. Chen, Z. Zhan, J. Zhang, J. Zhang, O. Liu, H. Liu, A genetic algorithm for the optimization of admission scheduling strategy in hospitals, in: G. Fogel, H. Ishibuchi (Eds.), Proceedings of the IEEE Congress on Evolutionary Computation (CEC) 2010, Barcelona, Spain, IEEE press, Piscataway, New Jersey, 1 – 5, 2010.

[3] R. J. Kusters, P. M. A. Groot, Modelling resource availability in general hospitals design and implementation of a decision support model, European Journal of Operational Research 88 (1996) 428 – 445.

[4] P. R. Harper, A framework for operational modelling of hospital resources, Health Care Management Science 5 (2002) 165 – 173.

[5] P. Jittamai, T. Kangwansura, A hospital admission planning model for emergency and elective patients under stochastic resource requirements and no-shows, in: The IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, 166–170, 2011.

[6] P. Demeester, W. Souffriau, P. D. Causmaecker, G. Vanden Berghe, A hybrid tabu search algorithm for automatically assigning patients to beds, Artificial Intelligence in Medicine 48 (2010) 61–70.

[7] S. Ceschia, A. Schaerf, Local search and lower bounds for the patient admission scheduling problem, Computers and Operations Research 38 (2011) 1452 – 1463.

[8] B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vanden Berghe, One hyper-heuristic approach to two timetabling problems in health care, Journal of Heuristics 18 (3) (2012) 401–434.

[9] T. Range, R. Lusby, J. Larsen, A column generation approach for solving the patient admission scheduling problem, European Journal of Operational Research 235 (2014) 252–264.

[10] W. Vancroonenburg, P. D. Causmaecker, G. V. Berghe, Patient-to-room assignment planning in a dynamic context, in: D. Kjenstad, A. Riise, T. E. Nordlander, B. McCollum, E. Burke (Eds.), Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, Son, Norway, SINTEF, Trondheim, Norway, 193 – 205, 2012.

[11] S. Ceschia, A. Schaerf, Modeling and solving the dynamic patient admission scheduling problem under uncertainty, Artificial Intelligence in Medicine 56 (2012) 199–205.

[12] S. Ceschia, A. Schaerf, Dynamic patient admission scheduling with operating room constraints, flexible horizons, and patient delays, Journal of Scheduling 19 (2016) 377–389.

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics 21 (6) (1953) 1087–1092.

[14] S. Kirkpatrick, D. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671 – 680.

[15] E. Aarts, J. Korst, W. Michiels, Simulated annealing, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (2005) 187–210.

[16] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, Lecture Notes in Computer Science 1520 (1998) 417 –431.

[17] D. Pisinger, S. Ropke, An adaptive large neighbourhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (2006) 455–472.

[18] R. S. Sutton, Reinforcement learning : An introduction, A Bradford Book, MIT Press, 1998.

[19] D. Meignan, A. Koukam, J.-C. Creput, Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism, Journal of Heuristics 16 (6) (2010) 859–879.

Appendix A. Notation

Table A.6 lists all introduced sets and subsets. Table A.7 lists all parameters for the described models. Table A.8 provides a list of all decision variables for the described models. Table A.9 summarizes the parameters for the algorithms described in the paper.

Set	Description
$\mathcal{P}$	$= \{1, \ldots, P\}$, the index set of all patients
$\mathcal{D}$	$= \{1, \ldots, D\}$, the index set of all days in the problem
$\mathcal{R}$	$= \{1, \ldots, R\}$, the index set of all rooms.
$\mathcal{P}_d$	$\subseteq \mathcal{P}$, the subset of patients which can be admitted on day $d \in \mathcal{D}$.
$\mathcal{D}_p$	$\subseteq \mathcal{D}$, the subset of days in which patient $p \in \mathcal{P}$ can be admitted into the hospital i.e. the potential set of first days at the hospital for the patient.
$\mathcal{R}_p$	$\subseteq \mathcal{R}$, the subset of rooms which are feasible for patient $p \in \mathcal{P}$
$\mathcal{R}^{SG}$	$\subseteq \mathcal{R}$, the subset of rooms which are gender segregated.
$\mathcal{F}$	$\subseteq \mathcal{P}$, the subset of patients which are female.
$\mathcal{M}$	$\subseteq \mathcal{P}$, the subset of patients which are male.

Table A.6: List of sets

Parameter	Description
d_p^{reg}	$\in \mathcal{D}$, the day on which patient p is registered in the hospital's administrative records.
d_p^{plan}	$\in \mathcal{D}$, the day on which patient p is expected to be admitted to the hospital.
d_p^{max}	$\in \mathcal{D}$, the last possible day patient p can be admitted.
l_p	the length of stay for patient $p \in \mathcal{P}$ corresponding to the number of nights the patient has to be admitted.
O_p	$\in \{0, 1\}$, overstay risk for patient $p \in \mathcal{P}$, which is equal to 1 if and only if patient p has a risk of staying one day longer than the length of stay l_p.
κ_r	the capacity (i.e. the number of beds) of room $r \in \mathcal{R}$.
c_{pr}	the daily cost of admitting the patient $p \in \mathcal{P}$ into the room $r \in \mathcal{R}$. If patient p and room r are not compatible then $c_{pr} = \infty$.
c^G	the cost per day a gender-segregated room (i.e. following the SG policy) includes patients of both genders.
c^D	the cost per day of delaying a patient.
c^O	the unit overcrowd cost for having potentially more patients in the room than the room's capacity.
c^T	the cost of a transfer.

Table A.7: List of model parameters

Variable	Description
x_{prd}	$\in \{0,1\}$, is equal to 1 if patient $p \in \mathcal{P}$ occupies a bed in room $r \in \mathcal{R}$ on the night following day $d \in \mathcal{D}$. The variable is fixed to zero for all days in which the patient cannot be present at the hospital i.e. $x_{prd} = 0$ if $r \notin \mathcal{R}_p$ or if the day d is either before d_p^{plan} or after $d_p^{\max} + l_p - 1$.
$\overline{x}_{prd}$	$\in \{0,1\}$, is equal to 1 if the patient $p \in \mathcal{P}$ may occupy a bed in room $r \in \mathcal{R}$ on the night following day $d \in \mathcal{D}$ after planned discharge (i.e. stay one day longer than anticipated) due to the overstay risk.
y_{pd}	$\in \{0,1\}$, is equal to 1 if patient $p \in \mathcal{P}$ is admitted into the hospital on day $d \in \mathcal{D}$. The variable is fixed to zero for all days in which it is not possible to admit patient p i.e. $y_{pd} = 0$ for all $d \notin \mathcal{D}_p$.
t_{pd}	$\in \{0,1\}$, is equal to one if patient $p \in \mathcal{P}$ is transferred from one room to another from day $d-1 \in \mathcal{D}$ to day $d \in \mathcal{D}$. For day 1 we define $t_{p1} = 0$.
f_{rd}	$\in \{0,1\}$, is equal to 1 if at least one female is occupying a bed in room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$.
m_{rd}	$\in \{0,1\}$, is equal to 1 if at least one male is occupying a bed in room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$.
b_{rd}	$\in \{0,1\}$, is equal to one if both genders are present in room $r \in \mathcal{R}$ on day $d \in \mathcal{D}$.
z_{rd}	≥ 0, measures the number of potential patients beyond the capacity, κ_r (i.e. the overcrowd), the room $r \in \mathcal{R}$ may have due to patients with overstay risk.

Table A.8: List of decision variables

Parameter	Description
A_d	$= \lvert\mathcal{P}_d\rvert$, the number of active patients on day $d \in \mathcal{D}$.
T	the current temperature in the SA.
T_{start}	the initial temperature of the SA.
T_{end}	the ending temperature for which the SA is terminated.
α	the cooling factor of the SA.
ρ_{acc}	the acceptance probability of the SA.
I	the index set of repair methods for the ALNS.
J	the index set of destroy methods for the ALNS.
H	$= I \cup J$ the joined set of repair and destroy methods for the ALNS.
w_h	the weight given to method $h \in H$.
π_h	the probability of choosing method $h \in H$.
s_h	the accumulated score given to method $h \in H$.
v_h	the number of times method $h \in H$ has been chosen.
σ	the additional score given to a method if chosen. It takes the value of either 0, σ_1, σ_2, and σ_3.
σ_1	if the new solution is the best observed so far.
σ_2	if the new solution is better than the previous solution.
σ_3	if the new solution is accepted but no better than the previous solution.
n_{iter}	the number of iterations over which the evaluation of the methods is performed (the weights w_h are recalculated each n_{iter} iteration).
γ	the reaction factor of the weights of the ALNS.
N	the number of patients removed and subsequently reinserted in ALNS.
ξ	is the percentage of active patients which can maximally be removed in an iteration of the ALNS.
$\overline{p}_j$	the j'th patient in a sorted list of patients where $j = 1, \ldots, A_d$ for $d \in \mathcal{D}$.
β	a randomization factor.
$\texttt{rel}_i(p_1, p_2)$	the i'th measure of relatedness between patients p_1 and p_2, $(i = 1, 2, 3, 4)$
ϱ_i	a value indicating the relative importance of relatedness measure i.
v_p^k	the regret cost of inserting patient $p \in \mathcal{P}$ in his/her k'best room compared to his/her best possible room.
$ntrials$	the number of trials performed for each temperature of the ALNS algorithm.
e	a parameter influencing the number of trials to perform ($ntrials = e \cdot R$).
n	the number of patients to remove in the Worst Removal destroy method.

Table A.9: List of algorithm parameters